Dealing with Categorical Variables - Lab

Introduction

In this lab, you'll explore the Boston Housing Data Set for categorical variables, and you'll transform your data so you'll be able to use categorical data as predictors!

Objectives

You will be able to:

- Identify and inspect the categorical variables in the Boston housing data set
- Learn how to categorize inputs that aren't categorical
- Create new datasets with dummy variables

Importing the Boston Housing data set

Let's start by importing the Boston Housing data set. This data set is available in Scikit-Learn, and can be imported running the column below.

In [1]:
```
import pandas as pd
from sklearn.datasets import load_boston
boston = load_boston()
```

If you'll inspect Boston now, you'll see that this basically returns a dictionary. Let's have a look at what exactly is stored in the dictionary by looking at the dictionary keys

In [2]:
```
print(boston)
```

```
{'data': array([[6.3200e-03, 1.8000e+01, 2.3100e+00, ..., 1.5300e+01, 3.9690e+02,
        4.9800e+00],
       [2.7310e-02, 0.0000e+00, 7.0700e+00, ..., 1.7800e+01, 3.9690e+02,
        9.1400e+00],
       [2.7290e-02, 0.0000e+00, 7.0700e+00, ..., 1.7800e+01, 3.9283e+02,
        4.0300e+00],
       ...,
       [6.0760e-02, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9690e+02,
        5.6400e+00],
       [1.0959e-01, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9345e+02,
        6.4800e+00],
       [4.7410e-02, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9690e+02,
        7.8800e+00]]), 'target': array([24. , 21.6, 34.7, 33.4, 36.2, 28.7, 22.9, 27.1, 16.5, 18.9, 15. ,
18.9, 21.7, 20.4, 18.2, 19.9, 23.1, 17.5, 20.2, 18.2, 13.6, 19.6,
15.2, 14.5, 15.6, 13.9, 16.6, 14.8, 18.4, 21. , 12.7, 14.5, 13.2,
13.1, 13.5, 18.9, 20. , 21. , 24.7, 30.8, 34.9, 26.6, 25.3, 24.7,
21.2, 19.3, 20. , 16.6, 14.4, 19.4, 19.7, 20.5, 25. , 23.4, 18.9,
35.4, 24.7, 31.6, 23.3, 19.6, 18.7, 16. , 22.2, 25. , 33. , 23.5,
19.4, 22. , 17.4, 20.9, 24.2, 21.7, 22.8, 23.4, 24.1, 21.4, 20. ,
20.8, 21.2, 20.3, 28. , 23.9, 24.8, 22.9, 23.9, 26.6, 22.5, 22.2,
23.6, 28.7, 22.6, 22. , 22.9, 25. , 20.6, 28.4, 21.4, 38.7, 43.8,
33.2, 27.5, 26.5, 18.6, 19.3, 20.1, 19.5, 19.5, 20.4, 19.8, 19.4,
21.7, 22.8, 18.8, 18.7, 18.5, 18.3, 21.2, 19.2, 20.4, 19.3, 22. ,
20.3, 20.5, 17.3, 18.8, 21.4, 15.7, 16.2, 18. , 14.3, 19.2, 19.6,
23. , 18.4, 15.6, 18.1, 17.4, 17.1, 13.3, 17.8, 14. , 14.4, 13.4,
15.6, 11.8, 13.8, 15.6, 14.6, 17.8, 15.4, 21.5, 19.6, 15.3, 19.4,
17. , 15.6, 13.1, 41.3, 24.3, 23.3, 27. , 50. , 50. , 50. , 22.7,
25. , 50. , 23.8, 23.8, 22.3, 17.4, 19.1, 23.1, 23.6, 22.6, 29.4,
23.2, 24.6, 29.9, 37.2, 39.8, 36.2, 37.9, 32.5, 26.4, 29.6, 50. ,
32. , 29.8, 34.9, 37. , 30.5, 36.4, 31.1, 29.1, 50. , 33.3, 30.3,
34.6, 34.9, 32.9, 24.1, 42.3, 48.5, 50. , 22.6, 24.4, 22.5, 24.4,
20. , 21.7, 19.3, 22.4, 28.1, 23.7, 25. , 23.3, 28.7, 21.5, 23. ,
26.7, 21.7, 27.5, 30.1, 44.8, 50. , 37.6, 31.6, 46.7, 31.5, 24.3,
31.7, 41.7, 48.3, 29. , 24. , 25.1, 31.5, 23.7, 23.3, 22. , 20.1,
22.2, 23.7, 17.6, 18.5, 24.3, 20.5, 24.5, 26.2, 24.4, 24.8, 29.6,
42.8, 21.9, 20.9, 44. , 50. , 36. , 30.1, 33.8, 43.1, 48.8, 31. ,
36.5, 22.8, 30.7, 50. , 43.5, 20.7, 21.1, 25.2, 24.4, 35.2, 32.4,
32. , 33.2, 33.1, 29.1, 35.1, 45.4, 35.4, 46. , 50. , 32.2, 22. ,
20.1, 23.2, 22.3, 24.8, 28.5, 37.3, 27.9, 23.9, 21.7, 28.6, 27.1,
20.3, 22.5, 29. , 24.8, 22. , 26.4, 33.1, 36.1, 28.4, 33.4, 28.2,
22.8, 20.3, 16.1, 22.1, 19.4, 21.6, 23.8, 16.2, 17.8, 19.8, 23.1,
21. , 23.8, 23.1, 20.4, 18.5, 25. , 24.6, 23. , 22.2, 19.3, 22.6,
19.8, 17.1, 19.4, 22.2, 20.7, 21.1, 19.5, 18.5, 20.6, 19. , 18.7,
32.7, 16.5, 23.9, 31.2, 17.5, 17.2, 23.1, 24.5, 26.6, 22.9, 24.1,
18.6, 30.1, 18.2, 20.6, 17.8, 21.7, 22.7, 22.6, 25. , 19.9, 20.8,
16.8, 21.9, 27.5, 21.9, 23.1, 50. , 50. , 50. , 50. , 50. , 13.8,
13.8, 15. , 13.9, 13.3, 13.1, 10.2, 10.4, 10.9, 11.3, 12.3,  8.8,
 7.2, 10.5,  7.4, 10.2, 11.5, 15.1, 23.2,  9.7, 13.8, 12.7, 13.1,
12.5,  8.5,  5. ,  6.3,  5.6,  7.2, 12.1,  8.3,  8.5,  5. , 11.9,
27.9, 17.2, 27.5, 15. , 17.2, 17.9, 16.3,  7. ,  7.2,  7.5, 10.4,
 8.8,  8.4, 16.7, 14.2, 20.8, 13.4, 11.7,  8.3, 10.2, 10.9, 11. ,
 9.5, 14.5, 14.1, 16.1, 14.3, 11.7, 13.4,  9.6,  8.7,  8.4, 12.8,
```

```
       10.5, 17.1, 18.4, 15.4, 10.8, 11.8, 14.9, 12.6, 14.1, 13. , 13.4,
       15.2, 16.1, 17.8, 14.9, 14.1, 12.7, 13.5, 14.9, 20. , 16.4, 17.7,
       19.5, 20.2, 21.4, 19.9, 19. , 19.1, 19.1, 20.1, 19.9, 19.6, 23.2,
       29.8, 13.8, 13.3, 16.7, 12. , 14.6, 21.4, 23. , 23.7, 25. , 21.8,
       20.6, 21.2, 19.1, 20.6, 15.2,  7. ,  8.1, 13.6, 20.1, 21.8, 24.5,
       23.1, 19.7, 18.3, 21.2, 17.5, 16.8, 22.4, 20.6, 23.9, 22. , 11.9]), 'feature_names': array(['CRIM',
'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD',
       'TAX', 'PTRATIO', 'B', 'LSTAT'], dtype='<U7'), 'DESCR': "Boston House Prices dataset\n==========
==============\n\nNotes\n------\nData Set Characteristics:  \n\n    :Number of Instances: 506 \n\n    :Num
ber of Attributes: 13 numeric/categorical predictive\n    \n    :Median Value (attribute 14) is usually th
e target\n\n    :Attribute Information (in order):\n        - CRIM     per capita crime rate by town\n
    - ZN       proportion of residential land zoned for lots over 25,000 sq.ft.\n        - INDUS    propor
tion of non-retail business acres per town\n        - CHAS     Charles River dummy variable (= 1 if tract
bounds river; 0 otherwise)\n        - NOX      nitric oxides concentration (parts per 10 million)\n
- RM       average number of rooms per dwelling\n        - AGE      proportion of owner-occupied units bui
lt prior to 1940\n        - DIS      weighted distances to five Boston employment centres\n        - RAD
    index of accessibility to radial highways\n        - TAX      full-value property-tax rate per $10,000
\n        - PTRATIO  pupil-teacher ratio by town\n        - B        1000(Bk - 0.63)^2 where Bk is the pro
portion of blacks by town\n        - LSTAT    % lower status of the population\n        - MEDV     Median
value of owner-occupied homes in $1000's\n\n    :Missing Attribute Values: None\n\n    :Creator: Harrison,
D. and Rubinfeld, D.L.\n\nThis is a copy of UCI ML housing dataset.\nhttp://archive.ics.uci.edu/ml/dataset
s/Housing\n\n\nThis dataset was taken from the StatLib library which is maintained at Carnegie Mellon Univ
ersity.\n\nThe Boston house-price data of Harrison, D. and Rubinfeld, D.L. 'Hedonic\nprices and the demand
for clean air', J. Environ. Economics & Management,\nvol.5, 81-102, 1978.   Used in Belsley, Kuh & Welsch,
'Regression diagnostics\n...', Wiley, 1980.   N.B. Various transformations are used in the table on\npages
244-261 of the latter.\n\nThe Boston house-price data has been used in many machine learning papers that a
ddress regression\nproblems.   \n     \n**References**\n\n   - Belsley, Kuh & Welsch, 'Regression diagnost
ics: Identifying Influential Data and Sources of Collinearity', Wiley, 1980. 244-261.\n   - Quinlan,R. (19
93). Combining Instance-Based and Model-Based Learning. In Proceedings on the Tenth International Conferen
ce of Machine Learning, 236-243, University of Massachusetts, Amherst. Morgan Kaufmann.\n   - many more!
(see http://archive.ics.uci.edu/ml/datasets/Housing)\n"}
```


In [3]:
```
boston.keys()
```

Out[3]:
```
dict_keys(['data', 'target', 'feature_names', 'DESCR'])
```

Let's create a Pandas DataFrame with the data (which are the features, not including the target) and the feature names as column names.

In [4]:
```
boston_features = pd.DataFrame(boston.data, columns = boston.feature_names)
```

In [5]:
```
boston_features.head()
```

Out[5]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33

For your reference, we copied the attribute information below. Additional information can be found here: http://scikit-learn.org/stable/datasets/index.html#boston-dataset

- CRIM: per capita crime rate by town
- ZN: proportion of residential land zoned for lots over 25,000 sq.ft.
- INDUS: proportion of non-retail business acres per town
- CHAS: Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
- NOX: nitric oxides concentration (parts per 10 million)
- RM: average number of rooms per dwelling
- AGE: proportion of owner-occupied units built prior to 1940
- DIS: weighted distances to five Boston employment centres
- RAD: index of accessibility to radial highways
- TAX: full-value property-tax rate per $10,000
- PTRATIO: pupil-teacher ratio by town
- B: 1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
- LSTAT: % lower status of the population

Let's convert the target to a dataframe as well, and assign the column name "MEDV"

In [6]:
```
boston_target = pd.DataFrame(boston.target, columns = ["MEDV"])
boston_target.head()
```

Out[6]:

	MEDV
0	24.0
1	21.6
2	34.7
3	33.4
4	36.2

The target is described as:

- MEDV: Median value of owner-occupied homes in $1000's

Next, let's merge the target and the predictors in one dataframe `boston_df`.

In [7]:
```
boston_df = pd.concat([boston_target, boston_features], axis=1)
boston_df.head()
```

Out[7]:

	MEDV	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	24.0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	21.6	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	34.7	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	33.4	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	36.2	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33

Let's inspect these 13 features using `.describe()` and `.info()`

In [8]:
```
boston_features.describe()
```

Out[8]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.0
mean	3.593761	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901	3.795043	9.549407	408.2
std	8.596783	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861	2.105710	8.707259	168.5
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.129600	1.000000	187.0
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000	2.100175	4.000000	279.0
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000	3.207450	5.000000	330.0
75%	3.647423	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000	5.188425	24.000000	666.0
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.126500	24.000000	711.0

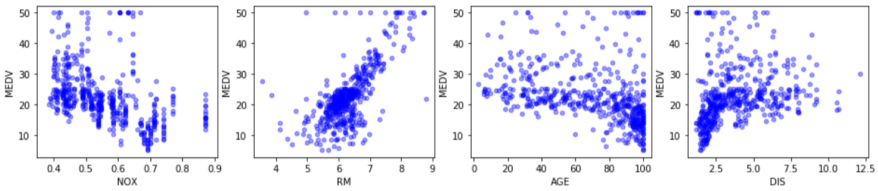
In [9]:
```
boston_features.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 13 columns):
CRIM       506 non-null float64
ZN         506 non-null float64
INDUS      506 non-null float64
CHAS       506 non-null float64
NOX        506 non-null float64
RM         506 non-null float64
AGE        506 non-null float64
DIS        506 non-null float64
RAD        506 non-null float64
TAX        506 non-null float64
PTRATIO    506 non-null float64
B          506 non-null float64
LSTAT      506 non-null float64
dtypes: float64(13)
memory usage: 51.5 KB
```

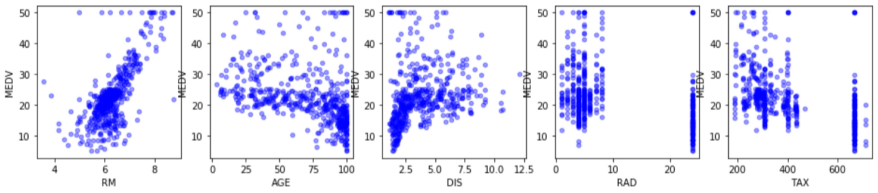
Now, take a look at the scatter plots for each predictor with the target on the y-axis.

In [10]:
```
import pandas as pd
import matplotlib.pyplot as plt

fig, axes = plt.subplots(nrows=1, ncols=4, figsize=(16,3))

for xcol, ax in zip(list(boston_features)[0:4], axes):
    boston_df.plot(kind='scatter', x= xcol, y="MEDV", ax=ax, alpha=0.4, color='b')
```

In [11]:
```
import pandas as pd
import matplotlib.pyplot as plt

fig, axes = plt.subplots(nrows=1, ncols=4, figsize=(16,3))

for xcol, ax in zip(list(boston_features)[4:8], axes):
    boston_df.plot(kind='scatter', x= xcol, y="MEDV", ax=ax, alpha=0.4, color='b')
```



In [12]:
```
import pandas as pd
import matplotlib.pyplot as plt

fig, axes = plt.subplots(nrows=1, ncols=5, figsize=(16,3))

for xcol, ax in zip(list(boston_features)[5:19], axes):
    boston_df.plot(kind='scatter', x= xcol, y="MEDV", ax=ax, alpha=0.4, color='b')
```



To categorical: binning

If you created your scatterplots correctly, you'll notice that except for CHAS (the Charles River Dummy variable), there is no clearly categorical data. You will have seen though that RAD and TAX have more of a vertical-looking structure like the one seen in the lesson, and that there is less of a "cloud"-looking structure compared to most other variables. It is difficult to justify a linear pattern between predictor and target here. In this situation, it might make sense to restructure data into bins so that they're treated as categorical variables. We'll start by showing how this can be done for RAD and then it's your turn to do this for TAX.

"RAD"

Look at the structure of "RAD" to decide how to create your bins.

In [13]:
```
boston_df["RAD"].describe()
```

Out[13]:
```
count    506.000000
mean       9.549407
std        8.707259
min        1.000000
25%        4.000000
50%        5.000000
75%       24.000000
max       24.000000
Name: RAD, dtype: float64
```

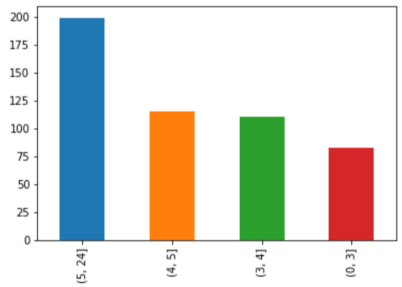
In [14]:
```
# first, create bins for based on the values observed. 5 values will result in 4 bins
bins = [0, 3, 4 , 5, 24]
# use pd.cut
bins_rad = pd.cut(boston_df['RAD'], bins)
```

In [15]:
```
# using pd.cut returns unordered categories. Transform this to ordered categories.
bins_rad = bins_rad.cat.as_unordered()
bins_rad.head()
```

Out[15]:
```
0    (0, 3]
1    (0, 3]
2    (0, 3]
3    (0, 3]
4    (0, 3]
Name: RAD, dtype: category
Categories (4, interval[int64]): [(0, 3], (3, 4], (4, 5], (5, 24]]
```

In [16]:
```
# inspect the result
bins_rad.value_counts().plot(kind='bar')
```

Out[16]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a16e0f860>
```



In [17]:
```
# replace the existing "RAD" column
boston_df["RAD"]=bins_rad
```

"TAX"

Split the "TAX" column up in 5 categories. You can chose the bins as desired but make sure they're pretty well-balanced.

In [18]:
```
boston_df["TAX"].describe()
```

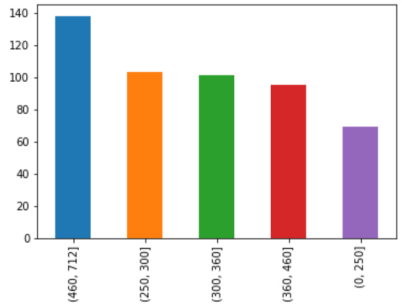
Out[18]:
```
count    506.000000
mean     408.237154
std      168.537116
min      187.000000
25%      279.000000
50%      330.000000
75%      666.000000
max      711.000000
Name: TAX, dtype: float64
```

In [19]:
```
# first, create bins for based on the values observed. 5 values will result in 4 bins
bins = [0, 250, 300, 360, 460, 712]
# use pd.cut
bins_tax = pd.cut(boston_df['TAX'], bins)
# using pd.cut returns unordered categories. Transform this to ordered categories.
bins_tax = bins_tax.cat.as_unordered()
bins_tax.head()
```

Out[19]:
```
0    (250, 300]
1      (0, 250]
2      (0, 250]
3      (0, 250]
4      (0, 250]
Name: TAX, dtype: category
Categories (5, interval[int64]): [(0, 250], (250, 300], (300, 360], (360, 460], (460, 712]]
```

In [20]:
```
# Check if the result is balanced
bins_tax.value_counts().plot(kind='bar')
```

Out[20]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a171c9320>
```



In [21]:
```
boston_df["TAX"]=bins_tax
```

Perform label encoding

In [22]:
```
boston_df["RAD"] = boston_df["RAD"].cat.codes
boston_df["TAX"] = boston_df["TAX"].cat.codes
```

In [23]:
```
boston_df.head()
```

Out[23]:

	MEDV	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	24.0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	0	1	15.3	396.90	4.98
1	21.6	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	0	0	17.8	396.90	9.14
2	34.7	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	0	0	17.8	392.83	4.03
3	33.4	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	0	0	18.7	394.63	2.94
4	36.2	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	0	0	18.7	396.90	5.33

Create dummy variables

Create dummy variables, and make sure their column names contain "TAX" and "RAD". Add the new dummy variables to boston_df and remove the old "RAD" and "TAX" columns.

In [24]:
```
tax_dummy = pd.get_dummies(bins_tax, prefix="TAX")
rad_dummy = pd.get_dummies(bins_rad, prefix="RAD")
```

In [25]:
```
boston_df = boston_df.drop(["RAD","TAX"], axis=1)
```

In [26]:
```
boston_df.head()
```

Out[26]:

	MEDV	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	PTRATIO	B	LSTAT
0	24.0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	15.3	396.90	4.98
1	21.6	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	17.8	396.90	9.14
2	34.7	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	17.8	392.83	4.03
3	33.4	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	18.7	394.63	2.94
4	36.2	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	18.7	396.90	5.33

In [27]:
```
boston_df = pd.concat([boston_df, rad_dummy, tax_dummy], axis=1)
boston_df.head()
```

Out[27]:

	MEDV	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	PTRATIO	...	LSTAT	RAD_(0, 3]	RAD_(3, 4]	RAD_(4, 5]	RAD_(5 24]
0	24.0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	15.3	...	4.98	1	0	0	0
1	21.6	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	17.8	...	9.14	1	0	0	0
2	34.7	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	17.8	...	4.03	1	0	0	0
3	33.4	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	18.7	...	2.94	1	0	0	0
4	36.2	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	18.7	...	5.33	1	0	0	0

5 rows × 21 columns

Note how you end up with 21 columns now!

Summary

In this lab, you practiced your categorical variable knowledge on the Boston Housing Data Set!